Efficient and Provably Secure Data Selective Sharing and Acquisition in Cloud-Based Systems

Kan Yang, *Senior Member, IEEE*, Jiangang Shu, *Member, IEEE*, and Ruitao Xie, *Member, IEEE*

***Abstract*— Towards the large amount of data generated everyday, data selective sharing and acquisition is one of the most significant data services in cloud-based systems, which enables data owners to selectively share their data to some particular users, and users to selectively acquire some interested data. However, it is challenging to protect data security and user privacy during data selective sharing and selective acquisition, because cloud servers are curious about the data or user's interests, and even send data to some unauthorized users or some uninterested users. In this paper, we propose an efficient and provably secure Data selective Sharing and Acquisition (DSA) scheme for cloud-based systems. Specifically, we first formulate a generic data selective sharing and acquisition problem in cloud-based systems by identifying several design goals in terms of correctness, soundness, security and efficiency. Then, we propose the DSA scheme to enable data owners to control the access of their data in a fine-grained manner, and enable users to refine the data acquisition without revealing their interests. Technically, a brand new cryptographic framework is developed to integrate attribute-based encryption with searchable encryption. Finally, we prove that the proposed DSA scheme is correct, sound, secure in the random oracle model, and efficient in practice.**

***Index Terms*— Cloud-based system, selective sharing, selective acquisition, access control, searchable encryption.**

I. Introduction

DUE to the economical, flexible and scalable storage and computing resources, cloud computing is becoming the most appropriate platform to store and process the ever-increasing amount of data generated every day [1]. Data sharing is one of the most fundamental services in cloud-based systems, where data owners rely on the cloud server to share data with other users. However, considering the confidentiality of data, data owners prefer to selectively share their data to some authorized users rather than all the users. On the other hand, considering the huge amount of data, users also want to selectively acquire some interested data instead of all the data shared by data owners. For example, existing healthcare cloud service providers (e.g., Amazon, Google, Microsoft, IBM, etc.) usually provide the service direct to the clinic or hospital, which means that the clinic or hospital will manage the healthcare data. However, it is a trend that patients are more willing to take control of their healthcare data as the data owners. Specifically, patients may selectively share their health data to certain types of users, e.g., doctors, health assessors in insurance companies or other patients suffering from the same health problems, meanwhile doctors (e.g., cardiologists) want to receive health data from selective patients (e.g., who have cardiovascular problems). However, public cloud service providers (e.g., Amazon, Google, Microsoft) cannot be fully trusted to enforce the data selective sharing and selective acquisition. Basically, there are two fundamental security and privacy requirements, namely: 1) *Data Confidentiality*: The shared data should not be known by the cloud server and any unauthorized users; and 2) *User Privacy*: The users' interests should not be known by the cloud server.

To selectively share their data, data owners define access policies of their data, but it is challenging to enforce these access policies because cloud servers are not fully trusted to evaluate access policies and make access decisions. A possible approach is to encrypt the shared data and only authorized users are given decryption keys. However, traditional public key encryption methods are not suitable for data encryption, because they usually produce multiple copies of ciphertexts for each data in the system, the number of which is proportional to the number of users. Alternatively, Attribute-Based Encryption (ABE) [2], [3], [4] is a good option for data encryption here, because: 1) it enables data owners to define fine-grained access policies over attributes; 2) access policies are enforced by cryptography rather than a trusted central server; and 3) it produces a single copy of ciphertexts regardless of the number of users. Based on ABE, many attribute-based access control schemes have been proposed for cloud storage systems with focus on access policy update [5] and attribute revocation [6].

However, the data encryption makes it difficult for data users to selectively acquire data by searching on the encrypted data. To cope with this problem, researchers propose to abstract a set of keywords from the data before being encrypted, and allow users to do the keyword search by providing a search trapdoor. However, the keywords will also reveal some private information of users to untrusted servers. For example, in a healthcare system, interest in psychological data reveals that the user or his/her relatives/friends may suffer from some

Manuscript received 13 March 2022; revised 29 June 2022 and 19 September 2022; accepted 10 October 2022. Date of publication 27 October 2022; date of current version 7 December 2022. This work was supported in part by NSF under Grant DGE-2146427 and in part by NSFC under Grant 62102204 and Grant 62272316. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Zekeriya Erkin. *(Corresponding authors: Kan Yang; Jiangang Shu.)*

Kan Yang is with the Department of Computer Science, University of Memphis, Memphis, TN 38152 USA (e-mail: kan.yang@memphis.edu).

Jiangang Shu is with the Peng Cheng National Laboratory, Department of New Networks, Shenzhen 518000, China (e-mail: jiangangshu@gmail.com).

Ruitao Xie is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: drtxie@gmail.com).

Digital Object Identifier 10.1109/TIFS.2022.3216956

kind of mental illness. To protect user privacy, Searchable Encryption (SE) schemes [7], [8], [9] have been developed to encrypt the keywords and enable users to generate encrypted trapdoors and search on encrypted keywords.

Individually, ABE and SE do a reasonably good job, but we have not yet developed the ability to simultaneously address data security and user privacy issues during data selective sharing and acquisition. To integrate ABE and SE, there are two approaches: *Policy-then-Keyword evaluation* and *Keyword-then-Policy evaluation*.

- *Policy-then-Keyword Evaluation* will first evaluate the access policies then evaluate the keyword matching. Under this framework, several attribute-based searchable encryption schemes [10], [11], [12], [13], [14], [15] have been developed by applying ABE to control the keyword search, where only the authorized users can do the keyword search. However, when applying ABE to encrypt the keyword, the output keyword index depends on the access policy. In particular, when two files have the same keyword but with different access policies, we cannot put two files on the same keyword file list due to different indices. The searching complexity for a single keyword is $\mathcal{O}(N_w \cdot N_p)$ where N_w is the total number of keywords and N_p is the max number of different access policies for any single keyword.
- *Keyword-then-Policy Evaluation* will first evaluate the keyword matching then evaluate the access policy. In such scenario, the access policy is associated with each file rather than just the keywords, so we can put different files under the file list of the same keyword regardless of their access policies. The searching complexity for a single keyword is $\mathcal{O}(N_w + N_p)$.

When a large number of access policies are associated to the files with the same keyword set, *Policy-then-Keyword Evaluation* is not as efficient as *Keyword-then-Policy Evaluation*. To achieve *Keyword-then-Policy Evaluation*, we cannot simply combine ABE and SE together (i.e., $SE(keyword)||ABE(data)$) [16], when the untrustworthy remote server can skip the keyword matching process and directly send data to those users who can decrypt successfully but have no interests at all. In practice, the server is highly incentive to conduct such behaviors, e.g., sends advertisement to users who have no interests. More specifically, the cloud server will launch the following bypass attacks: a) *Index Forging Attack*. It tries to forge the index, which is easy if the index can be generated by the public key; b) *Index Swapping Attack*. It swaps the indices of two ciphertexts; c) *Trapdoor Swapping Attack*. It uses the trapdoors of other users to match the index and deliver the data to users with no interests.

To resist these attacks, our idea is to bind the data and the index together, and tie the trapdoor and the transformed key together as well, such that the ciphertext can be decrypted correctly if and only if the index matches the trapdoor and the attributes of users satisfy the access policy. Specifically, we design a provably secure data selective sharing and acquisition (DSA) scheme, where data owners encrypt both data and keywords to obtain ciphertexts and indices; users generate a transformed secret key and a trapdoor for data query; the cloud server will first evaluate whether an index can match the trapdoor, if not move to the next index; when a keyword is matched, the cloud evaluates whether the access policy associated with the ciphertext can be satisfied by the attributes associated with the transformed secret key.

Considering that the cloud server will help users pre-decrypt data, the *indistinguishable security against chosen plaintext attack* (IND-CPA) is not sufficient for data security, because IND-CPA is only defined for the "passive" eavesdropping. In DSA scheme, we define a new data security model, called *selective and replayable indistinguishable security against chosen ciphertext attack* (*selective* IND-RCCA), which is identical to IND-CCA2 except for allowing the cloud server to generate new ciphertexts that decrypt to the same plaintext as a given ciphertext. We also define a relaxed version of *indistinguishable security against chosen keyword attack* (*selective* IND-CKA) for index security. Toward the trapdoor security, we define a weaker security model which only requires the trapdoor generation algorithm to be one-way (when given the trapdoor, it is hard to know the inside keyword).

In summary, as shown in Table I, the novelty of our DSA scheme includes: 1) our DSA scheme applies the keyword-then-policy evaluation framework so that the computation complexity is improved from $\mathcal{O}(N_w \cdot N_p)$ to $\mathcal{O}(N_w + N_p)$; 2) our DSA scheme can resist the keyword matching bypass attacks and achieve the soundness by binding the data and the index together and integrating the trapdoor and the transformed key together; 3) our DSA scheme enables the cloud server to evaluate both the keywords matching and attribute matching, and further help partially decrypt the data when both conditions are met, which can significantly reduce the computation cost on the users. The main contributions are summarized as follows.

1) We formulate a generic data selective sharing and acquisition problem in cloud-based systems and identify several design goals in terms of correctness, soundness, security and efficiency.
2) We propose the DSA scheme that enables: a) data owners to control the data sharing in a fine-grained way; b) users to refine the data acquisition without revealing their interests; and c) the cloud server to partially decrypt the data if the data is interesting to the user and the user has privileges to access the data.
3) We formally define the correctness, soundness and security of the DSA scheme, and prove that it is correct, sound, secure under the security models and random oracle model, and efficient in practice.

The remaining part of this paper is organized as follows. In Section II, we give the literature review on the data sharing and acquisition in cloud storage systems. Before describing the system model and design goals in Section III, we describe some preliminary definitions in Section IV. In Section V, we define the DSA scheme and its correctness, soundness and security. The detailed construction of DSA scheme is proposed in Section VI. The correctness and soundness are proved in Section VII, and the security proof and performance analysis are given in Section VIII. Then, we summarize the

TABLE I
COMPARISON WITH EXISTING SCHEMES

Scheme	Selective Sharing	Selective Acquisition	Soundness	Access Policy	Combination Approach	Searching Complexity	Main Idea
[10]–[15]	✓	✓	×	w	policy-then-keyword	$\mathcal{O}(N_w \cdot N_p)$	$SE(ABE(w))$
[16]	✓	✓	×	m	keyword-then-policy	$\mathcal{O}(N_w + N_p)$	$ABE(m) + SE(w)$
[17]	×	✓	✓	n/a	keyword-then-policy	$\mathcal{O}(N_w + N_p)$	$PKE(m) + PEKS(w) + MAC$
Our DSA	✓	✓	✓	m	keyword-then-policy	$\mathcal{O}(N_w + N_p)$	$SE(m) \rightarrow R \rightarrow ABE(m+R)$

m: data; w: keywords; R: randomness; N_w: the total number of keywords; N_p: max # of access policies for any single keyword.

paper in Section IX. In Appendix A and Appendix B, the detailed security proofs are described.

II. RELATED WORK

To protect the data from being seen by the cloud server or other unauthorized users, data are usually encrypted by the data owner before sending to the cloud. Attribute-based Encryption (ABE) [2], [3], [4] is a promising data encryption technique. There are two complementary forms of ABE, namely Key-Policy ABE (KP-ABE) [2] and Ciphertext-Policy ABE (CP-ABE) [3], [4]. Based on ABE, several attribute-based access control (ABAC) schemes [18], [19] have been proposed to ensure the data confidentiality in cloud-based systems. Specifically, ABAC allows data owners to define an access structure on attributes and encrypt the data under this access structure, such that data owners can define the attributes that the user needs to possess in order to decrypt the ciphertext.

To support the keyword search on encrypted data, Song et al. [7] proposed one of the first schemes for searching on encrypted data, which leverages symmetric key techniques and only allows the data encryptor to search. Boneh et al. [8] proposed Public Key Encryption with Keyword Search (PEKS), where anyone can encrypt the data by using the public key while only allow the owner of the corresponding secret key to search. To enable multiple users to search on the encrypted data, Hwang and Lee [20] proposed a multiuser public key encryption with conjunctive keyword search scheme. Camenisch et al. [21] proposed public key encryption with oblivious keyword search (PEOKS) where users can obtain trapdoors from the secret key holder without revealing the keywords. Li et al. [22] propose a multi-keyword ranked search on encrypted data by employing a secure K-nearest neighbors scheme.

Due to the advantages of ABE, attentions are paid to combine ABE with PEKS by constructing attribute-based encryption with keyword search (ABEKS) schemes [10], [11], [12], [13], [14], [15]. In [11], the authors propose an attribute-based proxy re-encryption scheme that can re-encrypt both the index and the trapdoor into the same key that can be evaluated by the cloud. In [23], an extended CP-ABE scheme is proposed to support single keyword search. However, the extension of keyword search (i.e., the query secret key sk_{query}) will break the security of the CP-ABE. It is easy to get the master secret key of the system $\mathsf{msk} = g^{\alpha}$ from the secret key of ABE sk_{abe} and the query secret key sk_{query} by calculating $\mathsf{sk}_{abe}/\mathsf{sk}_{query} = g^{\alpha} g^{at}/(g^{at} g^{u\alpha}) = (g^{\alpha})^{1-u}$, because u is

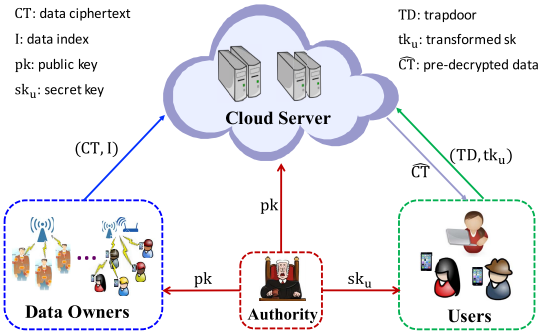


Fig. 1. Cloud-based data selective sharing and acquisition system.

selected by users. Once the master key is obtained, the user can decrypt all the ciphertexts regardless of his/her attributes. In [13] and [14], KP-ABE is employed to encrypt the keyword with a set of attributes and construct the trapdoor under access policies. In [10], [12], and [15], CP-ABE is used to encrypt the keywords. However, in data selective sharing and acquisition, anyone is allowed to search, and the access policy is defined on the data encryption not on the index encryption. Moreover, the soundness is not considered in previous works, where the cloud server can bypass the trapdoor and still deliver the data to users who do not have interests.

The soundness challenge was initially introduced by an index swap attack (swapping the indices of two messages) in [17] when people consider to combine Public Key Encryption (PKE) with Public Key Encryption with Keyword Search (PEKS). As mentioned in [17], a trivial solution is to simply append an authentication tag generated with a shared key between the data owner and data user. While it works, the solution destroys the asymmetric nature of public key encryption. The authors in [17] proposed a solution to combine PEKS with ElGamal based on the MAC produced with identity-based encryption [24]. However, the security requires the PKE is secure against plaintext checking attack (PCA). In [25], the authors propose a generic combination of PEKS and PKE, but still with an MAC. In this paper, we propose an integrate framework that internally ties the attribute-based encryption with searchable encryption, without needing any MAC.

III. SYSTEM MODEL AND DESIGN GOALS

A. System Model and Threat Model

We consider the cloud-based data selective sharing and acquisition system, as shown in Fig.1, which consists of four entities: data owners, the cloud server, users, and an authority.

1) Data Owners: Data owners are the owners of data and in most cases are also the producers of data. They will selectively share their data to users, such that users with different privileges have different views of the data. However, they do not trust the cloud server to control the sharing of their data. Hence, before sending data to the cloud server, data owners define an access policy for the shared data and encrypt them under this access policy. They also generate a set of indices for the shared data. Data owners are assumed to be fully trusted in the system.

2) Cloud Server: The cloud server stores the data and is responsible for evaluating whether the index can match the trapdoor and whether the user's attributes can satisfy the access policy. If both of these two conditions can be satisfied, the cloud server will help users pre-decrypt the data with the transformed secret key provided by the users together with the trapdoor. We assume that the cloud will follow the protocol to pre-decrypt and deliver those data which can satisfy both of the conditions, but it is also curious about the data shared by data owners. Moreover, we assume that the cloud server will try to send data to those unauthorized users or send data to those uninterested users.

3) Users: Each user holds a set of attributes that describe the role or identity of the user in the system. Accordingly, the user receives a secret key that is associated with the attributes. To selectively acquire his/her interested data, the user generates a trapdoor (used to filter the data) and a transformed his/her secret key (used to pre-decrypt the data), and send both of them to the cloud in one query. Users can collude with each other, but they will not send their secret keys to the cloud server or other adversaries.

4) Authority: The authority is the key management in the system. Specifically, it is responsible for managing the users' roles in the system by assigning different sets of attributes to them. According to the granted attributes of each user, it then issues a corresponding secret key to the user. It also publishes the public key that can be used for data encryption and index generation. We assume that the authority is fully trusted in the system, and there exists a secure communication channel between the authority and each user. We assume the authority will not collude with the cloud server or other adversary. For example, the trusted authority can be some agencies that are managed and audited by the government or other public organizations (similar to root DNS servers).

B. Design Goals

The ultimate goal of this paper is to design an efficient and provably secure data selective sharing and selective acquisition scheme for cloud-based systems, which allows data owners to selectively share their data and users to selectively acquire their interested data. Specifically, the scheme should achieve several design goals in terms of correctness, soundness, security and efficiency.

- *Correctness:* When access policy associated with the data can be satisfied by user's attributes, and the index of the data can match the trapdoor provided by the user, the data can be decrypted correctly by the user.
- *Soundness:* Users should not receive any data that cannot be decrypted or have no interests.
- *Security:* The data should be kept private against the cloud server or other unauthorized users. Neither the index nor the trapdoor should reveal user's interests.
- *Efficiency:* The scheme should not involve too much communication overhead and computation cost, especially for the computation cost on users who use mobile devices with limited resources to access data.

IV. Preliminaries

A. Linear Secret-Sharing Scheme (LSSS) Structure

Definition 1 (LSSS [26]): A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p$) if

1) The shares for each party form a vector over $\mathbb{Z}_p$.
2) There exists a matrix M called the share-generating matrix for Π. The matrix M has n rows and l columns. For all $i = 1, \cdots, n$, the i-th row of M is labeled by a party $\rho(i)$ (ρ is a function from $\{1, \cdots, n\}$ to $\mathcal{P}$). If the column vector $v = (s, r_2, \cdots, r_l)$ is considered, where $s \in \mathbb{Z}_p$ is the secret to be shared and $r_2, \cdots, r_l \in \mathbb{Z}_p$ are randomly chosen, then Mv is the vector of n shares of the secret s according to Π. The share $(Mv)_i$ belongs to party $\rho(i)$.

According to the above definition, the LSSS structure enjoys the *linear reconstruction* property: Suppose that Π is an LSSS for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized set, and let $I \subset \{1, 2, \cdots, n\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then, there exist constants $\{c_i \in \mathbb{Z}_p\}_{i \in I}$, s.t. for any valid shares $\{\lambda_i\}$ of a secret s according to Π, we have $\sum_{i \in I} c_i \lambda_i = s$. These constants $\{c_i\}$ can be found in polynomial time with the size of the share-generating matrix M, and for unauthorized sets, no such constants $\{c_i\}$ exist.

B. Bilinear Pairing

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three multiplicative groups with the same prime order p. A bilinear pairing is a mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

- **Bilinearity**: $e(u^a, v^b) = e(u, v)^{ab}$ for all $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$.
- **Non-degeneracy**: There exist $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ such that $e(u, v) \neq I$, where I is the identity element of $\mathbb{G}_T$.
- **Computability**: e can be efficiently computed.

C. Decisional q-Parallel Bilinear Diffie-Hellman Exponent Assumption

Definition 2 (Decisional q-parallel BDHE [4]): Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of order p, where $p > 2^\lambda$ is a prime. Suppose that there exists a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let $a, s, b_1, \cdots, b_q \in \mathbb{Z}_p$ be chosen randomly and g be a generator of $\mathbb{G}$. If an adversary is given by

$$\begin{aligned}\vec{y} = (&g, g^s, g^a, \cdots, g^{(a^q)}, \ , g^{(a^{q+2})}, \cdots, g^{(a^{2q})},\\ &\forall_{1 \le j \le q} \ \ g^{s \cdot b_j}, \ g^{a/b_j}, \cdots, g^{(a^q/b_j)}, \ ,\\ &g^{(a^{q+2}/b_j)}, \cdots, g^{(a^{2q}/b_j)},\\ &\forall_{1 \le j,k \le q, k \neq j} \ \ g^{a \cdot s \cdot b_k/b_j}, \cdots, g^{(a^q \cdot s \cdot b_k/b_j)}),\end{aligned}$$

it must be hard to distinguish a valid tuple $e(g,g)^{a^{q+1}s} \in \mathbb{G}_T$ from a random element R in $\mathbb{G}_T$.

An algorithm $\mathcal{B}$ that outputs $z \in \{0,1\}$ has advantage ϵ in solving q-parallel BDHE in $\mathbb{G}$ if

$$\left|\mathsf{Pr}[\mathcal{B}(\vec{y}, T = e(g,g)^{a^{q+1}s}) = 0] - \mathsf{Pr}[\mathcal{B}(\vec{y}, T = R) = 0]\right| \geq \epsilon.$$

Definition 3: The decisional q-parallel BDHE assumption holds if no polynomial time algorithm has a non-negligible advantage in solving the q-parallel BDHE problem. (It is generically secure as shown in Appendix C of [4].)

D. Decisional Linear (DLIN) Assumption

Definition 4 (DLIN): The challenger chooses a group $\mathbb{G}$ of prime order p. Let $a, b, x, y \in \mathbb{Z}_p$ be chosen randomly and g, h be generators of $\mathbb{G}$. When given $\vec{z} = (g, h, g^a, g^b, g^{ax}, g^{by})$, the adversary must distinguish a valid tuple $h^{x+y} \in \mathbb{G}$ from a random element R in $\mathbb{G}$. An algorithm $\mathcal{B}$ that outputs $z \in \{0,1\}$ has advantage ϵ in solving DLIN in $\mathbb{G}$ if

$$\left|\mathsf{Pr}[\mathcal{B}(\vec{z}, T = h^{x+y}) = 0] - \mathsf{Pr}[\mathcal{B}(\vec{z}, T = R) = 0]\right| \geq \epsilon.$$

Definition 5: The decisional linear (DLIN) assumption holds if no polynomial time algorithm has a non-negligible advantage in solving the DLIN problem.

V. Definitions

A. Definition of DSA

To meet all the requirements illustrated in Section III-B, we define the data selective sharing and acquisition scheme as

Definition 6 (DSA): A Data Selective Sharing and Acquisition scheme consists of the following algorithms:

- **Setup**$(1^\lambda) \rightarrow (\mathsf{msk}, \mathsf{pk})$. The setup algorithm takes the security parameter λ as input. It outputs the master secret key msk and the public key pk for the system.
- **SKGen**$(\mathsf{msk}, \mathsf{pk}, S_u) \rightarrow \mathsf{sk}_u$. The secret key generation algorithm takes as inputs the master secret key msk, the public key pk, and a set of attributes S_u assigned to each user u. It outputs a secret key sk_u for each user u.
- **Encrypt**$(\mathsf{pk}, (m, w), \mathbb{A}) \rightarrow (\mathsf{CT}, \mathsf{I})$. The encryption algorithm takes as inputs the public key pk, the data m from the data space $\mathcal{M}$ and a keyword w from the keyword space $\mathcal{W}$ describing the data,[1] and an access policy $\mathbb{A}$ defined over attributes. It consists of two subroutines:
  - $\mathsf{IndexGen}(\mathsf{pk}, w) \rightarrow (\mathsf{I}, \mathsf{R_I})$. The index generation subroutine outputs the data index I corresponding to the keyword w and a random index stamp R_I associated with this index. R_I will be used for encrypting each data in the keyword file list.
  - $\mathsf{DataEnc}(\mathsf{pk}, m, \mathbb{A}, \mathsf{R_I}) \rightarrow \mathsf{CT}$. The data encryption subroutine takes as input the random index stamp R_I and outputs a ciphertext CT.

  It outputs a tuple $(\mathsf{CT}, \mathsf{I})$, where CT is the data ciphertext and I is the index.
- **Query**$(\mathsf{sk}_u, \mathsf{pk}, w) \rightarrow (\mathsf{TD}, \mathsf{tk}_u, \hat{\mathsf{sk}}_u)$. The trapdoor generation algorithm takes as inputs the user's secret key sk_u, the public key pk and a keyword w describing his/her interests. It also consists of two subroutines:
  - $\mathsf{SKTran}(\mathsf{sk}_u) \rightarrow (\mathsf{tk}_u, \hat{\mathsf{sk}}_u)$. The secret key transformation subroutine outputs a transformed secret key tk_u and a decryption key $\hat{\mathsf{sk}}_u$.
  - $\mathsf{TDGen}(\mathsf{sk}_u, \hat{\mathsf{sk}}_u, \mathsf{pk}, w) \rightarrow \mathsf{TD}$. The trapdoor generation subroutine takes the decryption key $\hat{\mathsf{sk}}_u$ as one of its inputs, and generates the trapdoor TD associated with this decryption key.

  It outputs the trapdoor TD, the transformed secret key tk_u, and the corresponding decryption key $\hat{\mathsf{sk}}_u$.
- **Test**$(\mathsf{pk}, (\mathsf{TD}, \mathsf{I}), (\mathsf{tk}_u, \mathsf{CT}, \mathbb{A})) \rightarrow \hat{\mathsf{CT}}$ or $\perp$. The test algorithm takes as inputs the public key pk, the pair of trapdoor TD and the index I, the transformed secret key tk_u, the data ciphertext CT and its associated access policy $\mathbb{A}$. It also contains two testing subroutines:
  - $\mathsf{KTest}(\mathsf{pk}, \mathsf{TD}, \mathsf{I}) \rightarrow (R, Q)$ or $\perp$. The keyword test subroutine evaluates whether the keyword in the trapdoor TD can match the keyword in the index I. If they do not match, it terminates all the test algorithm and exits with a symbol $\perp$. Otherwise, the subroutine will recover the random element R used during the encryption, as well as another element Q containing the randomness of the index.
  - $\mathsf{ATest}(\mathsf{pk}, \mathsf{tk}_u, \mathsf{CT}, \mathbb{A}, R, Q) \rightarrow \hat{\mathsf{CT}}$ or $\perp$. The attribute test subroutine takes the random elements (R, Q) as part of inputs. It first evaluates whether the attributes contained in the transformed secret key tk_u can match the access policy $\mathbb{A}$ associated with the ciphertext CT. If they do not match, it terminates the test algorithm and exits with a symbol $\perp$. Otherwise, it outputs the pre-decrypted ciphertext $\hat{\mathsf{CT}}$.
- **Decrypt**$(\hat{\mathsf{sk}}_u, \hat{\mathsf{CT}}) \rightarrow m$. The decryption algorithm takes as inputs the decryption key $\hat{\mathsf{sk}}_u$ and the pre-decrypted ciphertext $\hat{\mathsf{CT}}$. It outputs the data m.

B. Definition of Correctness

The correctness of the scheme requires that if the keyword in the trapdoor can match the keyword in the index, and the user's attributes can satisfy the access policy associated with the data, then the user can finally decrypt the data successfully.

Definition 7 (Correctness): A data selective sharing and selective acquisition scheme DSA is correct, if $\forall \lambda \in \mathbb{N}$, and S_u satisfying $\mathbb{A}$, we have

$$\mathsf{Pr}[\mathsf{Decrypt}(\hat{\mathsf{sk}}_u, \mathsf{Test}(\mathsf{pk}, (\mathsf{TD}, \mathsf{I}), (\mathsf{tk}_u, \mathsf{CT}, \mathbb{A}))) = m] = 1,$$

where the probability is taken over the choice of

$$\begin{aligned}(\mathsf{msk}, \mathsf{pk}) &\leftarrow \mathsf{Setup}(1^\lambda),\\ \mathsf{sk}_u &\leftarrow \mathsf{SKGen}(\mathsf{msk}, \mathsf{pk}, S_u),\\ (\mathsf{CT}, \mathsf{I}) &\leftarrow \mathsf{Encrypt}(\mathsf{pk}, (m, w), \mathbb{A}),\\ (\mathsf{TD}, \mathsf{tk}_u, \hat{\mathsf{sk}}_u) &\leftarrow \mathsf{Query}(\mathsf{sk}_u, \mathsf{pk}, w).\end{aligned}$$

[1] For each keyword w in the keyword sets, we first run the $\mathsf{IndexGen}$ to generate a single index regardless of access policies. Then, we run the $\mathsf{DataEnc}$ algorithm to encrypt all the data in the file list according to their access policies. Without loss of generality, here we focus on a simple case with single keyword and single data.

C. Definition of Soundness

In DSA, we follow the *Keyword-then-Policy evaluation*. What happens if a malicious cloud directly sends the data to those users who can decrypt (policy match) but has no interests (keyword does not match). This cannot be covered by security. So, we define the soundness to cover three cases as follows.

- *Case 1 (Soundness of* KTest*)*: If the keyword inside the trapdoor does not match the keyword inside the index, then the probability to pass the KTest is negligible.
- *Case 2 (No Bypass of* KTest*)*: If the data can be decrypted but are not interested by this user, the cloud server should not be able to correctly pre-decrypt the data for this user.
- *Case 3 (Soundness of* Decrypt*)*: The pre-decrypted data is user-specific, which means that a user cannot successfully decrypt a pre-decrypted data generated for a different user with non-negligible probability.

D. Definitions of Security

We now give formal definitions of security for the data selective sharing and selective acquisition scheme.

1) Data Security: To prevent the cloud server and other adversaries from "passive" eavesdropping the data shared by the owners, the data encryption subroutine should be semantically secure against chosen plaintext attacks (IND-CPA). When considering the whole encryption algorithm (containing both the data encryption subroutine and the index generation subroutine), we employ a relaxed version of IND-CPA, called *selective* IND-CPA, where both of the two challenged plaintexts have the same keyword that does not equal to either challenged plaintext (i.e., (m_0, w), (m_1, w), and $w \neq m_0$, $w \neq m_1$). The keyword is selected in an initial phase at the very beginning of the security game.[2]

Besides launching the "passive" eavesdropping, the cloud server is also required to help users pre-decrypt the data ciphertext (i.e., transform the original data ciphertext to another data ciphertext that can be decrypted more easily). We employ a relaxed version of IND-CCA2 security (semantic security against adaptive chosen ciphertext attacks) defined in [27], called IND-Replayable CCA (IND-RCCA) security, which is identical to IND-CCA2 except for allowing the cloud server to generate new ciphertexts that decrypt to the same plaintext as a given ciphertext. This IND-RCCA model has also been applied in [28] for the decryption outsourcing of ABE. Following the similar definition, we define the *Selective* IND-RCCA Security for the data encryption through a Selective-IND-RCCA-Game.

*Definition 8 (***Selective-IND-RCCA-Game):** The Selective-IND-RCCA-Game is defined between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ whose running time is probabilistic polynomial in a security parameter λ as follows.

- **Init**: $\mathcal{A}$ gives the challenge access policy $\mathbb{A}^*$ and the challenge keyword w^* to $\mathcal{C}$.
- **Setup**: $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to generate $(\mathsf{msk}, \mathsf{pk})$, and gives pk to $\mathcal{A}$.
- **Phase 1**: $\mathcal{C}$ initializes an empty table T (recording all transformed secret key queries), an empty set D (recording the corrupted users), and an integer $j = 0$. $\mathcal{A}$ can adaptively make any of the following queries:
  - $\mathsf{TKQuery}(S_j)$: $\mathcal{A}$ queries the transformed key by giving a set of attributes S_j. $\mathcal{C}$ sets $j := j + 1$ and runs the SKGen to compute the corresponding secret key sk_j. $\mathcal{C}$ then transforms sk_j into the transformed secret key tk_j and the decryption key $\hat{\mathsf{sk}}_j$. $\mathcal{C}$ stores the entry $(j, S_j, \hat{\mathsf{sk}}_j, \mathsf{tk}_j)$ and returns tk_j to $\mathcal{A}$.
  - $\mathsf{DKQuery}(i)$: $\mathcal{A}$ cannot corrupt any key responding to the challenge access policy $\mathbb{A}^*$. If there exists an i^{th} entry in table T, $\mathcal{C}$ checks whether S_i can satisfy $\mathbb{A}^*$. If not, it sets $D := D \cup i$ and obtains the entry $(i, S_i, \hat{\mathsf{sk}}_i, \mathsf{tk}_i)$. Then, it returns the decryption key[3] $\hat{\mathsf{sk}}_i$ to $\mathcal{A}$. If no such entry exists or S_i can satisfy $\mathbb{A}^*$, it then returns $\perp$.
  - $\mathsf{Decrypt}(i, \mathsf{CT}, \mathbb{A})$: If there exists an i^{th} entry $(i, S_i, \hat{\mathsf{sk}}_i, \mathsf{tk}_i)$ in table T and S_i can satisfy $\mathbb{A}$, $\mathcal{C}$ decrypts the CT by running Test and Decrypt. Then, it returns the output to $\mathcal{A}$. Otherwise, it returns $\perp$.
- **Challenge**: $\mathcal{A}$ submits two equal-length messages m_0 and m_1, neither of which equals to the challenge keyword w^*. $\mathcal{C}$ flips a random coin τ, and encrypts m_τ under $\mathbb{A}^*$ and generates the index for keyword w^*. Then, both the ciphertext CT and the index I are given to $\mathcal{A}$.
- **Phase 2**: Phase 1 is repeated with the restriction that $\mathcal{A}$ cannot make a trivial decryption query, which means that Decrypt queries will be answered as in Phase 1, except that if the outputs would be either m_0 or m_1, then $\mathcal{C}$ responds with a special message **test** instead. [4]
- **Guess**: $\mathcal{A}$ outputs a guess τ' of τ.

We define $\mathcal{A}$'s advantage in Selective-IND-RCCA-Game by

$$\mathbf{Adv}_{DSA,\mathcal{A}}^{\text{Selective-IND-RCCA-Game}} = 2\mathsf{Pr}[\tau' = \tau] - 1.$$

2) Index Security: To protect the keyword security in the index, we require the adversary cannot distinguish two indices generated from two equal-length keywords, unless any corresponding trapdoor is revealed. We also require the index generation should be semantically secure against chosen keyword attacks (IND-CKA), where trapdoors can be queried adaptively. Considering that the index generation is only one subroutine of the encryption algorithm, we assume that both of the two challenge keywords are associated with the same challenge data. By selecting the challenge data in an initial phase, we define *Selective* IND-CKA security via the following Selective-IND-CKA-Game.

*Definition 9 (***Selective-IND-CKA-Game):** The Selective-IND-CKA-Game is defined between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ whose running time is probabilistic polynomial in a security parameter λ as follows.

- **Init**: $\mathcal{A}$ gives the challenge data m^* to $\mathcal{C}$.

[2]We can also define a stronger selective IND-CPA, where the keyword is selected during the challenge phase.

[3]The decryption key query has already implied secret key query, as the secret key can be constructed by the transformed key and the decryption key.

[4]Instead of comparing the ciphertext ($c = c^*$) in CCA game, this RCCA security game compares the decrypted plaintexts.

- **Setup**: $\mathcal{C}$ runs the $\mathsf{Setup}(1^\lambda)$ algorithm to generate $(\mathsf{msk}, \mathsf{pk})$. It gives pk to $\mathcal{A}$.
- **Phase 1**: $\mathcal{A}$ is allowed to query trapdoors for any keyword w_j.
- **Challenge**: $\mathcal{A}$ submits two equal-size keywords w_0, w_1. The only restriction is that neither w_0 nor w_1 has been queried in Phase 1. $\mathcal{C}$ first flips a random coin τ, and responses the index I_τ to $\mathcal{A}$ by running the Encrypt algorithm.
- **Phase 2**: Same as Phase 1 as long as the challenged keywords are not queried.
- **Guess**: $\mathcal{A}$ outputs a guess τ' of τ.

We define $\mathcal{A}$'s advantage in Selective-IND-CKA-Game by

$$\mathbf{Adv}_{DSA,\mathcal{A}}^{\text{Selective-IND-CKA-Game}} = 2\mathsf{Pr}[\tau' = \tau] - 1.$$

3) Trapdoor Security: As for the keyword privacy in the trapdoor, if the index is generated with private keys, then we can also define that the trapdoor should be semantically secure against chosen keyword attacks (IND-CKA). Similar to the index security, intuitively, the trapdoor security requires that the adversary is not able to distinguish two trapdoors of two equal-length keywords, unless any corresponding index is revealed.

However, if the index can be generated with public key, it is impossible to achieve the indistinguishable security against chosen keyword attacks (IND-CKA) unless the keyword space is sufficient large. This is called *offline keyword-guessing attack*, where the adversary can generate indices by guessing keywords, and use them to distinguish the two challenge trapdoors. Our DSA scheme is a public key encryption scheme where the index generation only needs the public key. Here, we define a weaker security model for trapdoor security, which only requires that the trapdoor security is one-way (when given the trapdoor, it is hard to know the inside keyword).

Definition 10 (Selective-IND-RCCA): A DSA scheme is Selective-IND-RCCA secure if all probabilistic polynomial-time adversaries have at most a negligible advantage in the above Selective-IND-RCCA-Game.

Definition 11 (Selective-IND-CKA): A DSA scheme is Selective-IND-CKA secure if all probabilistic polynomial-time adversaries have at most a negligible advantage in the above Selective-IND-CKA-Game.

*Definition 12 (*DSA *Security):* A DSA scheme is secure if it is Selective-IND-RCCA secure, Selective-IND-CKA secure and the trapdoor generation is one-way.

Remark 1: In the selective security definition, we assume that $w_0 = w_1$ or $m_0 = m_1$. We say that this security definition has already implied the case where $w_0 \neq w_1$ and $m_0 \neq m_1$:

- (m_0, w_0) IND (m_0, w_1): Selective IND-CKA for index (when adversary cannot get the tokens for W0 and W1)
- (m_0, w_1) IND (m_1, w_1): Selective IND-CPA for data (when adversary cannot get sufficient attributes)

Then, we can say: (m_0, w_0) IND (m_1, w_1).

VI. Construction of DSA

A. System Initialization by Authority

The authority initializes the system by running the setup algorithm as

Setup$(1^\lambda) \rightarrow (\mathsf{msk}, \mathsf{pk})$. It chooses two multiplicative groups $\mathbb{G}$ and $\mathbb{G}_T$ with the same prime order $p (p > 2^\lambda)$ and the bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ between them. Let g be a generator of $\mathbb{G}$. Let $H : \{0,1\}^* \rightarrow \mathbb{G}$ be the hash function that maps an arbitrary attribute to an element in group $\mathbb{G}$. It also defines a set of hash functions $H_0 : \{0,1\}^\lambda \rightarrow \mathbb{G}$, $H_1 : \mathbb{G}_T \rightarrow \{0,1\}^\lambda$, $H_2 : \mathbb{G}_T \rightarrow \{0,1\}^*$, $H_3 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, and $H_4 : \{0,1\}^* \rightarrow \{0,1\}^\lambda$. It chooses random numbers $\alpha,\ \beta,\ a,\ b,\ c \in \mathbb{Z}_p^*$ and sets the master secret key as $\mathsf{msk} = (\alpha,\ \beta,\ a,\ b,\ c)$. The public key is set as

$$\mathsf{pk} = (\ p, g, \mathbb{G}, \mathbb{G}_T, e, H, H_0, H_1, H_2, H_3, H_4, g^a, g^b, g^{abc}, g^\beta, e(g,g)^\alpha).$$

According to the attributes assigned to the data owner, the authority generates a corresponding secret key for the owner by running the secret key generation algorithm as

SKGen$(\mathsf{msk}, \mathsf{pk}, S_u) \rightarrow \mathsf{sk}_u$. For each user u who possesses the attribute set S_u, it chooses a random number $r_{sub} \in \mathbb{Z}_p^*$ and generates the secret key as

$$\mathsf{sk}_u = \Big(\ (ac, bc),\quad K = g^\alpha g^{\beta u},\quad K_a = g^{\frac{a}{a}} g^{\frac{\beta u}{a}},\quad K_b = g^{\frac{a}{b}} g^{\frac{\beta u}{b}},\ K' = g^u,\quad \forall x \in S_u :\ K_x = H(x)^u\Big).$$

where u is randomly chosen from $\mathbb{Z}_p^*$.

B. Data Encryption by Owners

To encrypt the data, the data owner first defines an access policy over attributes of users. In our construction, the access policy is described by an LSSS structure (M, ρ), where M is an $n \times l$ access matrix and ρ maps the rows of M to attributes. The data owner then runs the following encryption algorithm to encrypt the data m.[5] We follow the construction of [28] to transform the selective CPA secure CP-ABE to be RCCA secure.

Encrypt$(\mathsf{pk}, (m, w), (M, \rho)) \rightarrow (\mathsf{CT}, \mathsf{I})$. It consists of two subroutines:

- $\mathsf{IndexGen}(\mathsf{pk}, w) \rightarrow (\mathsf{I}, \mathsf{R_I})$. The index generation subroutine first selects a random string $R \leftarrow \{0,1\}^\lambda$ and two random numbers $s_1, s_2 \in \mathbb{Z}_p^*$ and sets $s_t = s_1 + s_2$. Then, it computes the ciphertext of the keyword w as

$$c_w = e\left(H_0(w), g^{abc}\right)^{s_t}.$$

Then, it outputs the index as

$$\mathsf{I} = \Big(I_1 = R \oplus H_1(c_w),\ I_2 = H_2(c_w),\ L_1 = (g^b)^{s_1},\quad L_2 = (g^a)^{s_2},\ L_3 = g^{s_t} \cdot H_0(w)^{s_t}\Big)$$

and the random index stamp $\mathsf{R_I} = (R, s_t)$.

[5] In real application, data m is first encrypted with a content key by using symmetric encryption methods. The content key is further encrypted by running the encryption algorithm Encrypt. For simplification, we directly use the data m.

- $\mathsf{DataEnc}(\mathsf{pk}, m, (M, \rho), \mathsf{R_I}) \rightarrow \mathsf{CT}$. Following the RCCA construction in [28], the data encryption subroutine first chooses a random string $k \leftarrow \{0,1\}^{\lambda}$ and computes $s_a = H_3(k, m)$ as the encryption secret. Then, it shares s_a through a random vector $\vec{v} = (s_a, y_2, \cdots, y_l)$. For $i = 1$ to n, it computes $\lambda_i = M_i \cdot \vec{v}$, where M_i is the vector corresponding to the i-th row of M. It takes as input the random index stamp (R, s_t) and outputs the ciphertext as

$$\mathsf{CT} = \Big(C_0 = m \oplus H_4(k), \quad C = k \cdot e(g,g)^{\alpha s_a},\ C' = g^{s_a + s_t} \cdot H_0(R), \quad \{C_i = g^{\beta\lambda_i} \cdot H(\rho(i))^{-r_i},\ D_i = g^{r_i}\}_{i\in[1,n]}\Big).$$

  where $r_1, \ldots, r_n$ are randomly chosen in $\mathbb{Z}_p^*$.

The data owner then uploads the data and its index to the cloud server in $\mathsf{I}||\mathsf{CT}||(M, \rho)$. Note that the access policy (M, ρ) is explicitly associated with the ciphertext.

C. Query Generation by Users

To generate a query for a keyword, the users will run the following algorithm:

Query$(\mathsf{sk}_u, \mathsf{pk}, w^*) \rightarrow (\mathsf{TD}, \mathsf{tk}_u, \hat{\mathsf{sk}}_u)$. It also consists of two subroutines:

- $\mathsf{SKTran}(\mathsf{sk}_u) \rightarrow (\mathsf{tk}_u, \hat{\mathsf{sk}}_u)$. The secret key transformation subroutine selects a random number $z \in \mathbb{Z}_p^*$ and transforms the user's secret key sk_u to a transformed secret key tk_u as

$$\mathsf{tk}_u = \Big(\hat{K} = (K)^z = g^{\alpha z} g^{\beta u z},\ \hat{K}' = (K')^z = g^{uz}, \quad \forall x \in S_u :\ \hat{K}_x = (K_x)^z = H(x)^{uz}\Big)$$

  and set the decryption key as $\hat{\mathsf{sk}}_u = z$.
- $\mathsf{TDGen}(\mathsf{sk}_u, \hat{\mathsf{sk}}_u, \mathsf{pk}, w) \rightarrow \mathsf{TD}$. The trapdoor generation subroutine takes the decryption key $\hat{\mathsf{sk}}_u$ as one of its inputs, and generates the trapdoor TD by randomly choosing $t_1, t_2 \in \mathbb{Z}_p^*$:

$$\mathsf{TD} = \Big(T_1 = g^{act_1} H_0(w^*)^{ac+act_1}, \quad \hat{T}_1 = (K_b)^z (g H_0(w^*))^{act_2}, \quad T_2 = g^{bct_1} H_0(w^*)^{bc+bct_1}, \quad \hat{T}_2 = (K_a)^z (g H_0(w^*))^{bct_2}, \quad T_3 = (g^{abc})^{t_1}, \hat{T}_3 = (g^{abc})^{t_2}\Big)$$

It sends the query $(\mathsf{TD}, \mathsf{tk}_u)$ to the cloud server, and keeps the corresponding decryption key $\hat{\mathsf{sk}}_u$.

D. Query Test by Cloud Server

Upon receiving the data and the query, the cloud server will test whether the keyword in the index can match the keyword in the trapdoor, and whether the attributes associated with the transformed secret key can match the access policy associated with the data ciphertext by running the test algorithm as

Test$(\mathsf{pk}, (\mathsf{TD}, \mathsf{I}), (\mathsf{tk}_u, \mathsf{CT}, \mathbb{A})) \rightarrow \hat{\mathsf{CT}}$ or $\perp$. The test algorithm also contains both two testing subroutines: keyword test subroutine and attribute test subroutine.

- $\mathsf{KTest}(\mathsf{pk}, \mathsf{TD}, \mathsf{I}) \rightarrow (R, Q)$ or $\perp$. The keyword test subroutine evaluates whether the keyword in the trapdoor TD can match the keyword in the index I. It first recovers the keyword ciphertext as

$$c_{w^*} = \frac{e(T_1, L_1) \cdot e(T_2, L_2)}{e(T_3, L_3)}. \tag{1}$$

  Then, it tests whether $H_2(c_{w^*}) \stackrel{?}{=} I_2$. If not, it aborts the test algorithm with a symbol $\perp$. Otherwise, we have $c_{w^*} = c_w$. Then, the subroutine will recover the random element R used during the encryption as $R = I_1 \oplus H_1(c_{w^*})$, and another element Q containing the randomness of the index

$$Q = \frac{e(\hat{T}_1, L_1) \cdot e(\hat{T}_2, L_2)}{e(\hat{T}_3, L_3)}. \tag{2}$$

  It outputs (R, Q).
- $\mathsf{ATest}(\mathsf{pk}, \mathsf{tk}_u, \mathsf{CT}, \mathbb{A}, R, Q) \rightarrow \hat{\mathsf{CT}}$ or $\perp$. The attribute test subroutine takes as inputs the random elements (R, Q). It first evaluates whether the attributes contained in the transformed secret key tk_u can match the access policy (M, ρ) associated with the ciphertext CT. If they do not match, it terminates the test algorithm and exits with a symbol $\perp$. Otherwise, the subroutine can find a set of constants $\{c_i\}$, s.t., $\sum_{i\in I} c_i M_i = (1, 0, \cdots, 0)$, where I is defined as $I = \{i : \rho(i) \in S_u\}$. Recall $\lambda_i = M_i \cdot \vec{v}$, we have $\sum_i c_i \lambda_i = s_a$. It then computes

$$P = \prod_{i\in I} \Big(e(C_i, \hat{K}') \cdot e(D_i, \hat{K}_{\rho(i)})\Big)^{c_i} \tag{3}$$

  and $\hat{C}' = C'/H_0(R)$. Finally, the subroutine computes

$$\hat{C} = \frac{e(\hat{C}', \hat{K})}{P \cdot Q} = e(g,g)^{\alpha z s_a} \tag{4}$$

  It outputs the pre-decrypted ciphertext as

$$\hat{\mathsf{CT}} = (C_0 = m \oplus H_4(k), \quad C = k \cdot e(g,g)^{\alpha s_a}, \quad \hat{C} = e(g,g)^{\alpha z s_a}).$$

E. Data Decryption by Users

Upon receiving the pre-decrypted data, the user can efficiently decrypt the data by running the decryption algorithm:

Decrypt$(\hat{\mathsf{sk}}_u, \hat{\mathsf{CT}}) \rightarrow m$. The decryption algorithm takes as inputs the decryption key $\hat{\mathsf{sk}}_u$ and the pre-decrypted ciphertext $\hat{\mathsf{CT}} = (C_0, C, \hat{C})$. It first computes

$$k = \frac{C}{\hat{C}^{\frac{1}{\hat{sk}_u}}} = \frac{k \cdot e(g,g)^{\alpha s_a}}{(e(g,g)^{\alpha z s_a})^{\frac{1}{z}}}.$$

Then, it recovers the data

$$m = C_0 \oplus H_4(k).$$

Now, it recomputes the $s' = H_3(k, m)$ and checks whether the following two equations can hold: $C \stackrel{?}{=} k \cdot e(g,g)^{\alpha s'}$ and $\hat{C} \stackrel{?}{=} e(g,g)^{\alpha s' z}$. If it does, the data m is accepted.

It is easy to find that the user only performs simple decryption computation, which is independent with the number of attributes in the ciphertext. The lightweight decryption algorithm can be easily implemented in many mobile devices with limited computation resources.

VII. Correctness and Soundness Proofs

A. Correctness Proof

Correctness of KTest: c_{w^*} can be computed as in Eq. 1. The details is shown in Eq. 5.
If the keyword in the trapdoor matches the keyword in the index, then

$$c_{w^*} = e(H_0(w^*), g)^{abcs_t} = e(H_0(w), g)^{abcs_t} = c_w$$

So, we have

$$H(c_{w^*}) = H(c_w).$$

Correctness of ATest: If $w^* = w$, Eq. 2 can be expressed as shown in Eq. 6.
and the Eq. 3 can be computed as

$$\begin{aligned} P &= \prod_{i\in I}\left(e(C_i, \hat{K}) \cdot e(D_i, \hat{K}_{\rho(i)})\right)^{c_i} \\ &= \prod_{i\in I}\left(e(g^{\beta\lambda_i} \cdot H(\rho(i))^{-r_i}, g^{uz}) \cdot e(g^{r_i}, H(\rho(i))^{uz})\right)^{c_i} \\ &= \prod_{i\in I} e(g,g)^{\beta uzc_i\lambda_i} \\ &= e(g,g)^{\beta uzs_a} \end{aligned}$$

Then, Eq. 4 can be verified as

$$\begin{aligned} \hat{C} &= \frac{e(\hat{C}', \hat{K})}{P \cdot Q} \\ &= \frac{e(g^{s_a} \cdot g^{s_t}, g^{\alpha z} g^{\beta uz})}{e(g,g)^{\beta uzs_a} \cdot e(g,g)^{\alpha zs_t} \cdot e(g,g)^{\beta uzs_t}} \\ &= \frac{e(g,g)^{\alpha zs_a} e(g,g)^{\alpha zs_t} \cdot e(g,g)^{\beta uzs_a} \cdot e(g,g)^{\beta uzs_t}}{e(g,g)^{\beta uzs_a} \cdot e(g,g)^{\alpha zs_t} \cdot e(g,g)^{\beta uzs_t}} \\ &= e(g,g)^{\alpha zs_a} \end{aligned}$$

B. Soundness Proof

Here, we show how to resist the bypass attacks defined in the soundness. We first prove the case 1 of soundness (soundness of KTest). Due to the collision resistance of hash functions, we say that two keywords $w^*, w \in \mathcal{W}, w^* \neq w$, the probability that these two keywords get the same hash values is negligible in λ. So, if $w^* \neq w$, the probability of passing KTest is also negligible.

For case 2 of soundness, suppose the index I_w can be matched by using another trapdoor $\mathsf{TD}_{u_2,w}$ generated by user u_2, then the cloud server can use this trapdoor to pass the KTest and get the keyword ciphertext c_w by Eq. 1. However, obtaining c_w does not help the pre-decryption, because the computation of component Q also requires the match of keywords in both index I_w and the trapdoor TD_{u_1,w^*}. From Eq. 2, we can see that if $H_0(w^*) \neq H_1(w)$, the component $e(H_0(w^*), g)^{abct_2s_t}$ and $e(H_0(w), g)^{abct_2s_t}$ cannot be canceled. Therefore, only when $H_0(w^*) = H_0(w), w^* \neq w$, Q can be recovered to $e(g,g)^{\beta uzs_t}$.

For case 3 of soundness, let's see what will happen if it uses the transformed secret key of user u_2 to compute the component Q. Note that both P and Q are closely related to u and z, which are user-specific. If the cloud server uses trapdoor and transformed secret key from another user u_2, then the pre-decrypted ciphertext cannot be decrypted correctly unless the decryption keys of both user u_1 and user u_2 are the same.

VIII. Security Proof and Performance Evaluation

A. Security Proof

We prove that the DSA scheme is secure in the random oracle model [29] by the following theorems:

Theorem 1: The DSA scheme is Selective-IND-RCCA secure in random oracle model, if the decisional q-parallel Bilinear Diffie-Hellman assumption holds.

Proof: The encryption algorithm in DSA scheme is constructed based on an adapted CP-ABE [4], which is proved to be selective CPA secure under the decisional q-parallel BDHE assumption. We also follow the construction of [28] to transform the selective CPA secure CP-ABE to be RCCA secure. Now, we prove that if there exists a polynomial time adversary $\mathcal{A}$ can play the Selective-IND-RCCA-Game with non-negligible advantage ϵ, we can build a simulator $\mathcal{B}$ that can break the selective CPA security of CP-ABE scheme in [4] with advantage ϵ plus a negligible amount. The detailed proof is described in Appendix A. ∎

Theorem 2: The DSA scheme is Selective-IND-CKA secure in random oracle model, if the decisional linear (DLIN) assumption holds.

Proof: We reduce the Selective-IND-CKA security of the DSA to the decisional linear (DLIN) assumption. That is, if there is a polynomial time adversary $\mathcal{A}$ can win the Selective-IND-CKA-Game with non-negligible advantage $\epsilon = \mathbf{Adv}^{\text{Selective-IND-RCCA-Game}}_{DSA,\mathcal{A}}$, we show how to build another polynomial time simulator $\mathcal{B}$ that plays the DLIN problem with non-negligible advantage. Please also refer to Appendix B for the detailed proof. ∎

Then, due to the hardness of discrete logarithm problem, it is easy to find that the trapdoor generation is one-way in polynomial time, thus we can say that the DSA scheme is secure in the random oracle model according to Definition 12.

B. Performance Analysis

In this section, we show the communication overhead between any two entities, and the computation cost on each entity. In order to show the communication overhead between any two entities in the DSA scheme, we first present the size of each component in the following table.

In real applications, the size of shared data is usually larger than the security parameter (e.g., 512 bits) in the elliptic curve. So, the data m are firstly encrypted with a content key by using symmetric encryption methods. The content key is further encrypted by running DSA encryption algorithm. In this case, the index size and trapdoor size only corresponding to the

TABLE II
SIZE OF EACH COMPONENT

Component	Size	example values
element size of $\mathbb{G}$	$\|p\|$	512 bits (α-curve)
user's attribute set S_u	$\|S_u\|$	5
attribute number of data m	$n_{att,m}$	5
keywords number of data m	$n_{kwd,m}$	5
public parameters pk	$16\|p\|$	8192 bits
user's secret key sk_u	$(\|S_u\|+5)\|p\|$	5120 bits
ciphertext CT	$(2n_{att,m}+3)\|p\|$	6656 bits
index I	$(3n_{kwd,m}+2)\|p\|$	8704 bits
transformed secret key tk_u	$(\|S_u\|+2)\|p\|$	3584 bits
trapdoor TD	$(4n_{kwd,m}+2)\|p\|$	11264 bits
partially decrypted data $\hat{\mathsf{CT}}$	$3\|p\|$	1536 bits

TABLE III
COMMUNICATION OVERHEAD

From \ To	Authority	Cloud Servers	Data Owners	Users
Authority	\	$\|\mathsf{pk}\|$	$\|\mathsf{pk}\|$	$\|\mathsf{sk}_u\|$
Cloud Servers	n/a	\	n/a	$\|\hat{\mathsf{CT}}\|$
Data Owners	n/a	$\|\mathsf{CT}\|+\|\mathsf{I}\|$	\	n/a
Users	n/a	$\|\mathsf{tk}_u\|+\|\mathsf{TD}\|$	n/a	\

number of keywords associated with the data. As shown in Table II, the index has only 8704 bits and the trapdoor is only 11264 bits long if we have 5 keywords associated with the data.

The communication cost of data ciphertexts are fixed from the data owner to the cloud server, and from the cloud server to the users. Here, we only evaluate the communication overhead of the designed DSA scheme which distributes the content keys. Based on Table II, we give a communication overhead analysis of the DSA scheme in Table III. We can see that the data sent from the cloud server to users are independent with the number of attributes it associated. This can reduce the communication overhead significantly when there are multiple data matching the trapdoor. The users only need to send the transformed secret key for one time whose size is linear to the number of attributes he/she holds, and receive data in constant size which are independent with the number of attributes associated with the data. The constant size data is actually the content key of symmetric key encryption.

To show the computation cost on each entity, we also do the experiments on a Unix system with an Intel Core i5 CPU at 2.4GHz and 8.00GB RAM. The code in the DSA scheme uses the Pairing-Based Cryptography (PBC) library version 0.5.12, and a symmetric elliptic curve α-curve, where the base field size is 512-bit and the embedding degree is 2. All the experimental results are the mean of 20 trials.

Fig. 2(a) shows the computation cost on data owners, which consists of both data encryption and index generation. The computation time of the data encryption is linear with the number of attributes involved in the access policy. To generate multiple indices for multiple keyword, the data owner can run the index generation algorithm multiple times and generate the index one by one. From Fig. 2(a), we can see that the data encryption algorithm is very efficient, even is more efficient than the index generation, when the number of keywords is the same as the number of attributes. This is because there is a pairing operation during each index generation.

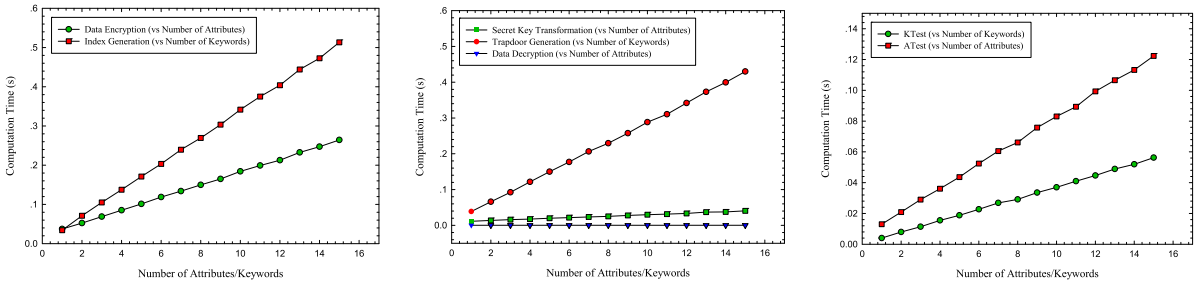
Fig. 2(b) shows the computation cost on users, which involves three operations: secret key transformation, trapdoor generation and the data decryption. We can see that both the secret key transformation and the data decryption only incur less computation cost. Note that, the data decryption here is independent with the number of attributes, and even is more efficient than transforming the simplest secret key which only consists one attribute. The trapdoor generation algorithm can also be run repeatedly when there are multiple keywords involved in the query.

Fig. 2(c) shows the keyword test KTest between the index and the trapdoor, and the access policy test ATest if passing the KTest. The ATest also help pre-decrypt data for users if the access policy can be satisfied by the attributes associated with the transformed secret key. We can see that the KTest is linear to the number of keywords and the ATest is also linear to the number of attributes. The cloud server does a major decryption computation, which significantly reduces the data decryption overhead on users.

From the above performance analysis, we can see that the main computation loads are outsourced from the user to the cloud by employing the predecryption on the cloud server. The communication cost and storage overhead are also small on

$$\begin{aligned}\frac{e(T_1,L_1)\cdot e(T_2,L_2)}{e(T_3,L_3)} &= \frac{e\left(g^{act_1}H_0(w^*)^{ac+act_1},(g^b)^{s_1}\right)e\left(g^{bct_1}H_0(w^*)^{bc+bct_1},(g^a)^{s_2}\right)}{e\left(g^{abct_1},g^{s_t}\cdot H_0(w)^{s_t}\right)}\\ &= \frac{e(g,g)^{abct_1(s_1+s_2)}e(H_0(w^*),g)^{abc(s_1+s_2)}e(H_0(w^*),g)^{abct_1(s_1+s_2)}}{e(g,g)^{abct_1s_t}e(H_0(w),g)^{abct_1s_t}}\end{aligned} \tag{5}$$

$$\begin{aligned}Q &= \frac{e(\hat{T}_1,L_1)\cdot e(\hat{T}_2,L_2)}{e(\hat{T}_3,L_3)}\\ &= \frac{e\left(g^{\frac{\alpha z+\beta uz}{b}}(gH_0(w^*))^{act_2},g^{bs_1}\right)e\left(g^{\frac{\alpha z+\beta uz}{a}}(gH_0(w^*))^{bct_2},g^{as_2}\right)}{e\left(g^{abct_2},g^{s_t}\cdot H_0(w)^{s_t}\right)}\\ &= \frac{e(g,g)^{\alpha zs_t}e(g,g)^{\beta uzs_t}e(g,g)^{abct_2s_t}e(H_0(w^*),g)^{abct_2s_t}}{e(g,g)^{abct_2s_t}e(H_0(w),g)^{abct_2s_t}}\\ &= e(g,g)^{\alpha zs_t}\cdot e(g,g)^{\beta uzs_t}\end{aligned} \tag{6}$$



(a) Data Owner (b) Users (c) Cloud Server

Fig. 2. Computation Cost.

the user side. Therefore, we can say that the DSA is efficient to be implemented in practice, especially on users' mobile devices whose computation capabilities or battery are usually limited.

IX. Conclusion

In this paper, we have studied the data selective sharing and selective acquisition problem in cloud-based systems. Specifically, we have first formulated the problem by identifying several design goals in terms of correctness, soundness, security and efficiency. Towards the security definition, we have proposed three new security models for data security, index security and trapdoor security, and further proposed an efficient and provably secure DSA scheme to enable data owners to control the sharing of their data in fine granularity and users to refine the data acquisition without revealing their interests. Finally, we have proved that the DSA is correct, sound, secure under the security models and random oracle model, and efficient in practice. In our future work, we are going to explore trapdoors with more complex predicate over keywords and extend to support multiple authorities.

Appendix A
Proof of Theorem 1

Theorem 1: The DSA *scheme is* Selective-IND-RCCA *secure in random oracle model, if the decisional q-parallel Bilinear Diffie-Hellman Exponent (Decisional q-parallel BDHE) assumption holds.*

Proof: The encryption algorithm in DSA scheme is constructed based on an adapted ciphertext-policy attribute-based encryption (CP-ABE) [4], which is proved to be selective CPA secure under the decisional q-parallel BDHE assumption. We also follow the construction of [28] to transform the selective CPA secure CP-ABE to be RCCA secure. Now, we prove that if there exists a polynomial time adversary $\mathcal{A}$ can play the Selective-IND-RCCA-Game with non-negligible advantage ϵ, we can build a simulator $\mathcal{B}$ that can break the selective CPA security of CP-ABE scheme in Appendix A of [4] with advantage ϵ plus a negligible amount. We use $\mathcal{C}_{abe}$ to denote the challenger in the security game of the CP-ABE scheme in Appendix A of [4].

Init: $\mathcal{A}$ chooses the challenge access policy (M^*, ρ^*) and the challenge keyword w^*, and sends them to the simulator $\mathcal{B}$. Then, $\mathcal{B}$ passes the challenge access policy (M^*, ρ^*) to $\mathcal{C}_{abe}$.

Setup: $\mathcal{B}$ obtains the CP-ABE public parameters $\mathsf{pk}_{abe} = (p, g, \mathbb{G}, \mathbb{G}_T, e, g^\beta, e(g,g)^\alpha)$ from $\mathcal{C}_{abe}$, and a description of hash function H. Then, it also randomly selects $a, b, c \in \mathbb{Z}_p^*$ and adds g^a, g^b, g^{abc} to pk_{abe} to generate the public key

$$\mathsf{pk} = (p, g, \mathbb{G}, \mathbb{G}_T, e, g^\beta, e(g,g)^\alpha, g^a, g^b, g^{abc}).$$

Then, $\mathcal{B}$ sends pk to the adversary $\mathcal{A}$.

Phase 1: $\mathcal{B}$ initializes empty tables $T, T_0, T_1, T_2, T_3, T_4$, an empty set D, and an integer $j = 0$. It answers the adversary $\mathcal{A}$'s queries as follows.

- *Random Oracle Hash* $H_0(w)$: If there is an entry $(w, H_0(w))$ in T_0, return $H_0(w)$. Otherwise, it chooses a random number $r_{0,w} \in \mathbb{Z}_p^*$ and sets $H_0(w) = g^{r_{0,w}}$. Then, it checks whether $g^{r_{0,w}}$ equals to any existing values, if so, it re-chooses another random number and checks again. Then, it records $(w, H_0(w) = g^{r_{0,w}})$ in T_0 and return $H_0(w)$.
- *Random Oracle Hash* $H_1(c_w)$: If there is an entry $(c_w, H_1(c_w))$ in T_1, return $H_1(c_w)$. Otherwise, it chooses a random number $r_{1,c_w} \in \{0,1\}^\lambda$ and sets $H_1(c_w) = r_{1,c_w}$. Then, it checks whether r_{1,c_w} equals to any existing values, if so, it re-chooses another random number and checks again. Then, it records $(c_w, H_1(c_w) = r_{1,c_w})$ in T_1 and return $H_1(c_w)$.
- *Random Oracle Hash* $H_2(c_w)$: If there is an entry $(c_w, H_2(c_w))$ in T_2, return $H_2(c_w)$. Otherwise, it chooses a random number $r_{2,c_w} \in \{0,1\}^*$ and sets $H_2(c_w) = r_{2,c_w}$. Then, it checks whether r_{2,c_w} equals to any existing values, if so, it re-chooses another random number and checks again. Then, it records $(c_w, H_2(c_w) = r_{2,c_w})$ in T_2 and return $H_2(c_w)$.
- *Random Oracle Hash* $H_3(k, m)$: If there is an entry $(k, m, H_3(k, m))$ in T_3, return $H_3(k, m)$. Otherwise, it chooses a random number $r_{3,k,m} \in \mathbb{Z}_p^*$ and sets $H_3(k, m) = r_{3,k,m}$. Then, it checks whether $r_{3,k,m}$ equals to any existing values, if so, it re-chooses another random number and checks again. Then, it records $(k, m, H_3(k, m) = r_{3,k,m})$ in T_3 and return $H_3(k, m)$.
- *Random Oracle Hash* $H_4(k)$: If there is an entry $(k, H_4(k))$ in T_4, return $H_4(k)$. Otherwise, it chooses a random number $r_{4,k} \in \{0,1\}^\lambda$ and sets $H_4(k) = r_{4,k}$. Then, it checks whether $r_{4,k}$ equals to any existing values, if so, it re-chooses another random number and checks again. Then, it records $(k, H_4(k) = r_{4,k})$ in T_4 and return $H_4(k)$.
- $\mathsf{TKQuery}(S_j)$: Basically, $\mathcal{A}$ is able to query any transformed key by giving a set of attributes S_j. $\mathcal{B}$ sets $j := j + 1$. Considering that the adversary is not allowed to

query the secret key that can satisfy the challenge access policy (M^*, ρ^*) in the security model of the CP-ABE scheme in [4], but the adversary in our security model is able to query the transformed secret key for any set of attributes. Thus, the simulator $\mathcal{B}$ generates two different types of transformed secret keys:

- If S_j satisfies (M^*, ρ^*), the simulator $\mathcal{B}$ is not able to receive the correct secret key corresponding to S_j from the challenger $\mathcal{C}_{abe}$. Thus, the simulator $\mathcal{B}$ generates a "special" transformed secret key as follows: it chooses a random $\gamma \in \mathbb{Z}_p^*$ as the "master key", and generates a secret key sk'. Then, it sets the transformed secret key as

$$\mathsf{tk}_j = (K = g^{\gamma} g^{\beta u_j}, K' = g^{u_j}, \forall x \in S_j : K_x = H(x)^{u_j}).$$

  and sets the decryption key $\hat{\mathsf{sk}}_j = \gamma$.
- If S_j does not satisfy (M^*, ρ^*), the simulator $\mathcal{B}$ obtains the secret key $\mathsf{sk}_{abe,j}$ from the challenger $\mathcal{C}_{abe}$,

$$\mathsf{sk}_{abe,j} = (K = g^{\alpha} g^{\beta u_j}, K' = g^{u_j}, \forall x \in S_j : K_x = H(x)^{u_j}),$$

  and generates the transformed secret key as

$$\mathsf{tk}_j = (\hat{K} = g^{\alpha z_j} g^{\beta u_j z_j}, \hat{K}' = g^{u_j z_j}, \forall x \in S_j : \hat{K}_x = H(x)^{u_j z_j})$$

  by choosing a random number $z_j \in \mathbb{Z}_p^*$ as the decryption key $\hat{\mathsf{sk}}_j$.

Finally, $\mathcal{B}$ stores the entry $(j, S_j, \hat{\mathsf{sk}}_j, \mathsf{tk}_j)$ in Table T and returns tk_j to $\mathcal{A}$.

- $\mathsf{DKQuery}(i)$: $\mathcal{A}$ cannot corrupt any key responding to the challenge access policy $\mathbb{A}^*$. If there exists an i^{th} entry in table T and S_i does not satisfy (M^*, ρ^*), it sets $D := D \cup i$ and obtains the entry $(i, S_i, \hat{\mathsf{sk}}_i, \mathsf{tk}_i)$. Then, it returns the decryption key $\hat{\mathsf{sk}}_i$ to $\mathcal{A}$. If no such entry exists or S_i can satisfy $\mathbb{A}^*$, it then returns $\perp$.
- $\mathsf{Decrypt}(i, \mathsf{CT}, \mathbb{A})$: Considering that the adversary $\mathcal{A}$ can obtain all the transformed secret keys for any sets of attributes, and it can also query the trapdoor TD_{w^*} corresponding to the challenge keyword w^*, the adversary is able to obtain any pre-decrypted data. Thus, without loss of generality, we assume that all the ciphertexts input to this $\mathsf{Decrypt}$ oracle are already pre-decrypted to the ElGamal form of

$$\hat{\mathsf{CT}} = (C_0 = m \oplus H_4(k),\ C = k \cdot e(g,g)^{\alpha s_a}, \hat{C} = e(g,g)^{\alpha z s_a}),$$

  which is associated with an access policy (M, ρ). It then obtains the i^{th} entry $(i, S_i, \hat{\mathsf{sk}}_i, \mathsf{tk}_i)$ in table T, where S_i can satisfy (M, ρ). If there is no such entry in table T_1, it returns $\perp$. Then, it checks whether S_i can satisfy the challenge access policy (M^*, ρ^*) and proceeds as follows:
  - If $S_i \in (M^*, \rho^*)$, then the pre-decrypted data should be in the form as

$$\hat{\mathsf{CT}} = (C_0 = m \oplus H_4(k), C = k \cdot e(g,g)^{\alpha s}, \hat{C} = e(g,g)^{\gamma s})$$

    Then, it computes $\phi = \hat{C}^{1/\gamma} = e(g,g)^s$. For each entry $(R_{0,i},\ m_i,\ s_i = H_3(R_{0,i}, m_i))$ in T_3, the simulator $\mathcal{B}$ checks if $e(g,g)^{s_i} \stackrel{?}{=} \phi$. If no entry matches, it outputs $\perp$ to $\mathcal{A}$. If more than one matches are found, it aborts the simulation. Suppose $(k,\ m,\ s = H_3(k,m))$ is the only match, it gets $H_4(k)$ and tests if $C_0 = m \oplus H_4(k)$, $C = k \cdot e(g,g)^{\alpha s}$, and $\hat{C} = e(g,g)^{\gamma s}$. If all tests are passed, it sends m to the adversary $\mathcal{A}$. Otherwise, it returns $\perp$ to the adversary.
  - If $S_i \notin (M^*, \rho^*)$, then the pre-decrypted data should be in the normal form as

$$\hat{\mathsf{CT}} = (C_0 = m \oplus H_4(k), C = k \cdot e(g,g)^{\alpha s}, \hat{C} = e(g,g)^{\alpha s z}).$$

    Then, it computes $k = \frac{C}{\hat{C}^{1/z}}$ and obtains the record $(k,\ m,\ s = H_3(k,m))$ from table T_3. If there is no such record existing in T_3, it aborts with $\perp$. It then gets $H_4(k)$ and tests if $C_0 = m \oplus H_4(k)$, $C = k \cdot e(g,g)^{\alpha s}$, and $\hat{C} = e(g,g)^{\alpha s z}$. If all tests are passed, it sends m to the adversary $\mathcal{A}$. Otherwise, it returns $\perp$ to the adversary.

Challenge: $\mathcal{A}$ submits two equal-length messages m_0 and m_1, neither of which equals to the challenge keyword w^*. $\mathcal{C}$ then chooses a random string $r \in \{0,1\}^{\lambda}$ and sets $R_{0,0} = m_0 \oplus r$ and $R_{0,1} = m_1 \oplus r$. It passes $R_{0,0}$ and $R_{0,1}$ to the challenger $\mathcal{C}_{abe}$. $\mathcal{C}_{abe}$ then flips a random coin τ and outputs a ciphertext $\mathsf{CT}_{abe,\tau}$ associated with (M^*, ρ^*) as

$$\mathsf{CT}_{abe,\tau} = (C_\tau,\ C'_{abe},\ \{C_i, D_i\}_{i \in [1,n]}).$$

$\mathcal{B}$ then choose two random strings $C_0, R \in \{0,1\}^{\lambda}$ and a random number $s_t \in \mathbb{Z}_p^*$, and obtains the challenge ciphertext

$$\mathsf{CT}_{\tau} = (C_0,\ C_\tau,\ C' = C'_{abe} \cdot g^{s_t} \cdot H_0(R),\ \{C_i, D_i\}_{i \in [1,n]})$$

To simulate the index, the simulator $\mathcal{B}$ first computes the ciphertext of the challenge keyword w^* as

$$c_{w^*} = e\left(H_0(w^*), g^{abc}\right)^{s_t}.$$

It then randomly chooses $s_1 \in \mathbb{Z}_p^*$ and sets $s_2 = s_t - s_1$. Then, it outputs the index as

$$\mathsf{I}_{w^*} = \Big(I_1 = R \oplus H_1(c_{w^*}),\ I_2 = H_2(c_{w^*}), L_1 = (g^b)^{s_1},\ L_2 = (g^a)^{s_2},\ L_3 = g^{s_t} \cdot H_0(c_{w^*})^{s_t}\Big)$$

The simulator sends both $(\mathsf{CT}_\tau, \mathsf{I}_{w^*})$ to the adversary $\mathcal{A}$.

Phase 2: Phase 1 is repeated with the restriction that $\mathcal{A}$ cannot make a trivial decryption query, which means that Decrypt queries will be answered as in Phase 1, except that if the outputs would be either m_0 or m_1, then $\mathcal{B}$ responds with a special message **test** instead.

Guess: $\mathcal{A}$ outputs a guess τ' of τ or abort. Now, the simulator $\mathcal{B}$ searches through the table T_3 to see if the values $R_{0,0}$, $R_{0,1}$, m_0 or m_1 appears in any entry. If neither or both values appear, $\mathcal{B}$ outputs the same τ' as its guess. If only value $R_{0,\tau''}$ appears, $\mathcal{B}$ just ignores the $\mathcal{A}$'s guess and outputs τ'' as its guess.

The advantage of the simulator in the CP-ABE game can be computed as

$$\mathcal{A}_{\mathcal{B}}^{\text{Selective-IND-CPA}} = \frac{1}{2}\cdot\epsilon + \frac{1}{2}\cdot\epsilon_0,$$

where ϵ is the advantage of the adversary $\mathcal{A}$ in our security game and ϵ_0 is the probability that there is an entry $(R_b, m_b, s_a = H_3(R_b, m_b))$ in the table T_3 and passes all the following test: $C' = g^{s_a}$ and $C_b = R_b \cdot e(g,g)^{\alpha s_a}$. However, this probability is negligible.

Therefore, we can say that if ϵ is non-negligible, $\mathcal{A}_{\mathcal{B}}^{\text{Selective-IND-CPA}}$ is also non-negligible, which breaks the CP-ABE scheme in [4]. ∎

APPENDIX B
PROOF OF THEOREM 2

Theorem 2: The DSA *scheme is* Selective-IND-CKA *secure in random oracle model, if the decisional linear (DLIN) assumption holds. Proof:* We reduce the Selective-IND-CKA security of the DSA to the decisional linear (DLIN) assumption. That is, if there is a polynomial time adversary $\mathcal{A}$ can win the Selective-IND-CKA-Game with non-negligible advantage $\epsilon = \mathbf{Adv}_{DSA,\mathcal{A}}^{\text{Selective-IND-RCCA-Game}}$, we show how to build another polynomial time simulator $\mathcal{B}$ that plays the DLIN problem with non-negligible advantage.

Init: $\mathcal{A}$ selects a challenge data m^* from the data space and sends it to $\mathcal{B}$.

Setup: $\mathcal{B}$ runs the $\mathsf{Setup}(1^\lambda)$ algorithm to generate

$$\mathsf{msk} = (\alpha,\ \beta,\ a,\ b,\ c).$$

The public key is set as

$$\mathsf{pk} = \Big(\ p, g, \mathbb{G}, \mathbb{G}_T, e, g^a, g^b, g^{abc}, g^\beta, e(g,g)^\alpha\ \Big).$$

It gives pk to $\mathcal{A}$.

Phase 1: $\mathcal{A}$ queries trapdoors for any keyword w_j and transformed secret keys for any set of attributes S_j. To simulate the trapdoor generation algorithm, the simulator $\mathcal{B}$ first simulates the random oracle of $H_0(w)$ by maintaining a table T_0. For a new keyword w not appeared in the table, it chooses a random number $r_{0,w} \in \mathbb{Z}_p^*$ and sets $H_0(w) = g^{r_{0,w}}$, where $H_0(w)$ should not equal to any existing values. Otherwise, it re-chooses another random number and checks again. Then, it stores this entry $(w, H_0(w))$ into the table T_0. If there is already an entry corresponding to the queried keyword w, it just returns the existing value $H_0(w)$. Similarly, it use another table T to simulate the random oracle of $H(x)$ for any attribute x by setting $H(x) = g^{r_x}$, where $r_x \in \mathbb{Z}_p^*$ is randomly selected. Moreover, it also simulates the random oracles of H_1 and H_2 as

- *Random Oracle Hash* $H_1(c_w)$: If there is an entry $(c_w, H_1(c_w))$ in T_1, return $H_1(c_w)$. Otherwise, it chooses a random number $r_{1,c_w} \in \{0,1\}^\lambda$ and sets $H_1(c_w) = r_{1,c_w}$. Then, it checks whether r_{1,c_w} equals to any existing values, if so, it re-chooses another random number and checks again. Then, it records $(c_w, H_1(c_w) = r_{1,c_w})$ in T_1 and return $H_1(c_w)$.
- *Random Oracle Hash* $H_2(c_w)$: If there is an entry $(c_w, H_2(c_w))$ in T_2, return $H_2(c_w)$. Otherwise, it chooses a random number $r_{2,c_w} \in \{0,1\}^*$ and sets $H_2(c_w) = r_{2,c_w}$. Then, it checks whether r_{2,c_w} equals to any existing values, if so, it re-chooses another random number and checks again. Then, it records $(c_w, H_2(c_w) = r_{2,c_w})$ in T_2 and return $H_2(c_w)$.

It first gets a secret key by querying the secret key generation algorithm SKGen for the attribute set S_j as

$$\mathsf{sk}_u = \Big((ac, bc),\ K = g^\alpha g^{\beta u},\ K_a = g^{\frac{a}{a}} g^{\frac{\beta u}{a}},\ K_b = g^{\frac{a}{b}} g^{\frac{\beta u}{b}},\ K' = g^u,\ \forall x \in S_u :\ K_x = H(x)^u\ \Big).$$

Note that, the secret key can be first queried from the secret key generation oracle in the underlying CP-ABE scheme [4] and then generates K_a and K_b from K, which means that the CP-ABE can be used as a blackbox.

Then, it transforms the secret key to compute a transformed secret key tk_u as

$$\mathsf{tk}_u = \Big(\ \hat{K} = (K)^z = g^{\alpha z} g^{\beta u z},\ \hat{K}' = (K')^z = g^{uz},\ \forall x \in S_u :\ \hat{K}_x = (K_x)^z = H(x)^{uz}\ \Big)$$

and sends tk_u to the adversary $\mathcal{A}$, but keeps the decryption key as $\hat{\mathsf{sk}}_u = z$.

Note that, the adversary $\mathcal{A}$ can query any transformed secret key corresponding to any sets of attributes in this game. Actually, the challenge data is selected by the adversary, and the ciphertext of the challenge data is independent with the keyword. Therefore, we can say that the data ciphertext and the transformed secret keys do not increase the advantage of distinguishing the index.

The simulator takes the decryption key and the secret key as input, and runs TDGen to generate the trapdoor of w_j as

$$\mathsf{TD}_j = \Big(T_{1,j} = g^{act_{1,j}} H_0(w_j)^{ac+act_{1,j}},\ \hat{T}_{1,j} = (K_b)^z (g H_0(w_j))^{act_{2,j}},\ T_{2,j} = g^{bct_1} H_0(w_j)^{bc+bct_1},\ \hat{T}_{2,j} = (K_a)^z (g H_0(w_j))^{bct_{2,j}},\ T_{3,j} = (g^{abc})^{t_{1,j}},\ \hat{T}_{3,j} = (g^{abc})^{t_{2,j}}\Big)$$

Challenge: $\mathcal{A}$ submits two equal-size keywords w_0, w_1. The only restriction is that neither w_0 nor w_1 has been queried in Phase 1. $\mathcal{C}$ first flips a random coin τ, and runs the Encrypt algorithm by taking (m^*, w_τ) as inputs. Because the ciphertext will not increase the advantage of distinguishing the index, so we omit the ciphertext here. Suppose the random element associated with the ciphertext is R. The simulator takes this random element R as one of inputs and computes the ciphertext of the keyword w_b as

$$c_{w_\tau} = e\left(H_0(w_\tau), g^{abc}\right)^{s_t}.$$

It then randomly chooses $s_1 \in \mathbb{Z}_p^*$ and set $s_2 = s_t - s_1$. Then, it outputs the index as

$$\mathsf{I}_{w_\tau} = \Big(I_1 = R \oplus H_1(c_{w_\tau}),\ I_2 = H_2(c_{w_\tau}),$$
$$L_1 = (g^b)^{s_1},\ L_2 = (g^a)^{s_2},\ L_3 = g^{s_t} \cdot H_0(w_\tau)^{s_t}\ \Big)$$

Phase 2: Same as Phase 1 as long as the challenged keywords are not queried.

Guess: $\mathcal{A}$ outputs a guess τ' of τ.

Because the random element R is uniformly distributed, $I_1 = R \oplus H_1(c_{w_\tau})$ is also uniformly distributed. Moreover, L_1, L_2 is also uniformly distributed. If there is an adversary $\mathcal{A}$ wins the above Selective-IND-CKA-Game with non-negligible advantage, we can say that when given $(g^a, g^b, g^{bs_1}, g^{as_2})$, the adversary can distinguish the tuple $(h = g \cdot H_0(w_\tau), h^{s_1+s_2} = (g \cdot H_0(w_\tau))^{s_t})$ from a (h, R) (where R is a random element in $\mathbb{G}$) with non-negligible advantage, which contradicts the DLIN assumption. ■

REFERENCES

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. 800-145, 2011.

[2] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS)*, 2006, pp. 89–98.

[3] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 321–334.

[4] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Public Key Cryptography*. Berlin, Germany: Springer-Verlag, 2011, pp. 53–70.

[5] K. Yang, X. Jia, K. Ren, R. Xie, and L. Huang, "Enabling efficient access control with dynamic policy updating for big data in the cloud," in *Proc. INFOCOM*, Apr. 2014, pp. 2013–2021.

[6] K. Yang, X. Jia, K. Ren, B. Zhang, and R. Xie, "DAC-MACS: Effective data access control for multiauthority cloud storage systems," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 11, pp. 1790–1801, Nov. 2013.

[7] D. Xiaoding Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy. (SP)*, 2000, pp. 44–55.

[8] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. EUROCRYPT*, in Lecture Notes in Computer Science, vol. 3027. Springer, 2004, pp. 506–522.

[9] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory Cryptography*, in Lecture Notes in Computer Science, vol. 4392. Springer, 2007, pp. 535–554.

[10] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2014, pp. 226–234.

[11] K. Liang and W. Susilo, "Searchable attribute-based mechanism with efficient data sharing for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 9, pp. 1981–1992, Sep. 2015.

[12] J. Shi, J. Lai, Y. Li, R. H. Deng, and J. Weng, "Authorized keyword search on encrypted data," in *Proc. Eur. Symp. Res. Comput. Secur.*, in Lecture Notes in Computer Science, vol. 8712. Springer, 2014, pp. 419–435.

[13] H. Cui, Z. Wan, R. Deng, G. Wang, and Y. Li, "Efficient and expressive keyword search over encrypted data in cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 409–422, May/Jun. 2018.

[14] Y. Yu, J. Shi, H. Li, Y. Li, X. Du, and M. Guizani, "Key-policy attribute-based encryption with keyword search in virtualized environments," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1242–1251, Jun. 2020.

[15] H. Yin et al., "CP-ABSE: A ciphertext-policy attribute-based searchable encryption scheme," *IEEE Access*, vol. 7, pp. 5682–5694, 2019.

[16] A. Michalas, "The lord of the shares: Combining attribute-based encryption and searchable encryption for flexible data sharing," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput.*, Apr. 2019, pp. 146–155.

[17] J. Baek, R. Safavi-Naini, and W. Susilo, "On the integration of public key data encryption and public key encryption with keyword search," in *Proc. Int. Conf. Inf. Secur.*, in Lecture Notes in Computer Science, vol. 4176. Springer, 2006, pp. 217–232.

[18] K. Yang, X. Jia, and K. Ren, "Secure and verifiable policy update outsourcing for big data access control in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3461–3470, Dec. 2015.

[19] K. Yang, Z. Liu, X. Jia, and X. S. Shen, "Time-domain attribute-based access control for cloud-based video content sharing: A cryptographic approach," *IEEE Trans. Multimedia*, vol. 18, no. 5, pp. 940–950, May 2016.

[20] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Proc. Int. Conf. Pairing-Based Cryptogr.*, in Lecture Notes in Computer Science, vol. 4575. Springer, 2007, pp. 2–22.

[21] J. Camenisch, M. Kohlweiss, A. Rial, and C. Sheedy, "Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data," in *Proc. Int. Workshop Public Key Cryptogr.*, in Lecture Notes in Computer Science, vol. 5443. Springer, 2009, pp. 196–214.

[22] H. Li, D. Liu, Y. Dai, T. H. Luan, and X. S. Shen, "Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 1, pp. 127–138, Mar. 2015.

[23] C. Wang, W. Li, Y. Li, and X. Xu, "A ciphertext-policy attribute-based encryption scheme supporting keyword search function," in *Cyberspace Safety and Security*, in Lecture Notes in Computer Science, vol. 8300. Springer, 2013, pp. 377–386.

[24] D. Boneh and M. K. Franklin, "Identity-based encryption from the Weil pairing," in *Proc. CRYPTO*. London, U.K.: Springer-Verlag, 2001, pp. 213–229.

[25] R. Zhang and H. Imai, "Generic combination of public key encryption with keyword search and public key encryption," in *Proc. Int. Conf. Cryptol. Netw. Secur.*, in Lecture Notes in Computer Science, vol. 4856. Springer, 2007, pp. 159–174.

[26] A. Beimel, "Secure schemes for secret sharing and key distribution," Ph.D. dissertation, Israel Inst. Technol., Technion, Haifa, Israel, 1996.

[27] R. Canetti, H. Krawczyk, and J. B. Nielsen, "Relaxing chosen-ciphertext security," in *Advances in Cryptology*, in Lecture Notes in Computer Science, vol. 2729. Springer, 2003, pp. 565–582.

[28] M. Green, S. Hohenberger, and B. Waters, "Outsourcing the decryption of ABE ciphertexts," in *Proc. 20th USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2011, p. 34.

[29] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. 1st ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 1993, pp. 62–73.

Kan Yang (Senior Member, IEEE) received the B.Eng. degree in information security from the University of Science and Technology of China in 2008 and the Ph.D. degree in computer science from the City University of Hong Kong in 2013. He is currently an Assistant Professor with the Department of Computer Science, University of Memphis, USA. His research interests include data security, blockchain, AI security, network security, and applied cryptography.

Jiangang Shu (Member, IEEE) received the Ph.D. degree in computer science from the City University of Hong Kong (CityU), Hong Kong, in 2019. He is currently a Research Scientist with the Peng Cheng National Laboratory, Department of New Networks, Shenzhen, China. He was a Postgraduate Visiting Student with the Secure Mobile Centre, Singapore Management School, Singapore, and a Post-Doctoral Fellow at CityU. His research interests include AI privacy, crowdsourcing and cloud security, the IoT security, applied cryptography, data security and privacy, and searchable encryption.

Ruitao Xie (Member, IEEE) received the B.Eng. degree from the Beijing University of Posts and Telecommunications in 2008 and the Ph.D. degree in computer science from the City University of Hong Kong in 2014. She is currently an Assistant Professor with the College of Computer Science and Software Engineering, Shenzhen University. Her research interests include AI networking and mobile computing, distributed systems, and cloud computing.